NIS Part II Report

Discussing the *boulder-chat* protocol

Jonah Hooper
University of Cape Town

Yusri Dollie
University of Cape Town

William Grant
University of Cape Town

1 INTRODUCTION

boulder-Chat is an encrypted chat protocol and command-line client developed in Python. The program itself runs via an HTTP Server created using the Flask micro framework and uses both RSA for asymmetric and AES for symmetric key encryption. All hashing is done using SHA256.

2 CLIENT AUTHENTICATION

Given 2 principals A and B who are two parties who wish to communicate over the *boulder-chat* protocol. Assuming that this is the **first time** that A attempts to communicate with B then A must query the authentication server S in order to authenticate itself with B. A will send the following information to S when attempting to authenticate: K_A that is A's public key; K_B that is B's public key. Once S receives this information it will return a payload that contains: $(\#K_{AB})K_S$ that is the hashed session/symmetric key signed by S for the communication session between A and B; $(\#(K_A \mid\mid K_B))K_S$ a hashed version of both A and Bs public keys that is signed by S which acts as proof of the authentication of A; $K_{AB}\}K_A$ which is the symmetric/session key encrypted by A's public key.

Message 1 A -> S: K_A, K_B
Message 2 S -> A: $(\#K_{AB})K_S$, $(\#(K_A \mid\mid K_B))K_S$, $K_{AB}\}K_A$

3 CLIENT COMMUNICATION

3.1 First Time Communication - key exchange

A will send the following information to B on the first communication $\{\#K_{AB}\}K_B$ that is an signed version of the session/symmetric key encrypted by B's public key; $(K_AK_B)K_S$ that is an authorization token of A and B's public keys signed by S's master key. This proves that A is authorized for communication by S; $(\#M)K_A$ that is a hashed version of the message that A wishes to send that is signed by the by the public key of A. Further, $(M)K_{AB}$ - the first message A wishes to send to B is encrypted using the session/symmetric key.

Message 3: A-> B: $\{K_{AB}\}K_B$, $(K_AK_B)K_S$, $(\#M)K_A$, $(M)K_{AB}$

3.2 General Communication

Going forward when A wishes to send a message to B it will send through the following information: $(\#M)K_A$ that is the hashed message M signed using A's public key and the actual message M, encrypted with $\{M\}K_{AB}$.

Message 4: A-> B: $\{\#M\}K_A$, $\{M\}K_{AB}$

3.3 File Transfer

It is also possible to send files via the HTTP server using end-to-end encryption through boulder-chat with the power of the Flask micro framework.

4 DESIGN MOTIVATION

The *boulder-chat* protocol is was designed to be a secure chat protocol and was created with various security mechanisms to prevent security attacks such as data modification (manipulation), interception and fabrication.

Given principals A and B where A wishes to communicate with B, in order to prevent attacks and tampering during the authentication of a A by the server S the session/symmetric key is hashed using SHA256 and then signed using the secret key of S. This ensures that the symmetric key is not tampered with in transit. Further the symmetric key generated by S is encrypted using the public key of A. This ensures that only A can read the message meaning that non-repudiation and is enforced and interception is computationally expensive. A concatenation of the A and B public keys is signed by the server. This is then sent to A as an authentication token which is used later to authenticate the identity of A with B. This enables B to know with certainty that the sender of the message is actually A and not someone else and that A is authorized by the server.

The exchange of symmetric keys between A and B is secure. A signs a hash of the unencrypted message and sends it along with the message encrypted by the symmetric key both encrypted with B's public key. This ensures that if the symmetric key or message are tampered with in transit the resulting unencrypted message will not match the signature. It would be computationally expensive to tamper with all 3 at the same time. In addition, the use of B's public key to encrypt the symmetric key ensures non-repudiation since only B could decrypt the correct symmetric key and thus decrypt the message. The symmetric key encryption of the message also ensures that a message cannot be intercepted since only someone with B's public key can decrypt the symmetric key. B can use the authentication token (concatenation of A and B's public keys signed by S) as proof that A was authenticated by S.

Once they symmetric key exchange has occurred there is an implicit trust between A and B. When A or B wish to communicate they merely have to sign the messages of the hashes they wish to send and encrypt the messages public keys. The signing ensures that there can be no tampering with the messages. It also ensures that the message cannot be fabricated. The symmetric key encryption prevents interception attacks. The key exchange creates implicit trust between A and B which means that only A or B can read a message sent by the other principle and so the protocol supports non-repudiation.